MONTE-CARLO METHODS MAKE DEMPSTER-SHAFER FORMALISM FEASIBLE

Vladik Kreinovich, Andrew Bernat, Walter Borrett,
Yvonne Mariscal, Elsa Villa
Computer Science Department
The University of Texas at El Paso
El Paso, TX 79968, USA

Abstract: One of the main obstacles to the applications of Dempster-Shafer formalism is its computational complexity. If we combine m different pieces of knowledge, then in general case we have to perform up to 2^m computational steps, which for large m is infeasible. For several important cases algorithms with smaller running time have been proposed. We prove, however, that if we want to compute the belief $bel(Q)$ in any given query Q, then exponential time is inevitable.

It is still inevitable, if we want to compute $bel(Q)$ with given precision ϵ. This restriction corresponds to the natural idea that since initial masses are known only approximately, there is no sense in trying to compute $bel(Q)$ precisely. A further idea is that there is always some doubt in the whole knowledge, so there is always a probability p_0 that the expert's knowledge is wrong. In view of that it is sufficient to have an algorithm that gives a correct answer a probability $> 1 - p_0$. If we use the original Dempster's combination rule, this possibility diminishes the running time, but still leaves the problem infeasible in the general case.

We show that for the alternative combination rules proposed by Smets and Yager feasible methods exist. We also show how these methods can be parallelized, and what parallelization model fits this problem best.

Keywords: Dempster-Shafer formalism, combination rules, Monte-Carlo methods, feasible, parallel.

1. FORMULATION OF THE PROBLEM

Dempster-Shafer formalism in brief. Dempster-Shafer (DS) formalism, proposed in (Shafer, 1976), is very promising and is already widely used. In this formalism knowledge is described by a finite set of statements $E_1, ..., E_n$, to each of which a number (*mass*) $m(E_i)$ is assigned so that $\sum_i m(E_i) = 1$. Then, if someone asks a query Q, we must

(11-2)

produce as an answer the *belief* $bel(Q)$ that Q is true. This belief is defined as a sum of masses $m(E_i)$ of all the statements E_i that imply Q (i.e., for which $E_i \to Q$ is true). In addition to belief in Q one can also ask for the *plausibility* $pl(Q)$ of Q.

Comment. From the computational viewpoint the problems of computing bel and pl are equivalent, because $bel(Q) = 1 - pl(\neg Q)$ and $pl(Q) = 1 - bel(\neg Q)$. Therefore in the following text we'll analyze only the problem of computing beliefs. In (Yager, 1987) a slightly different definition of plausibility is given; our negative theorems and algorithms can be easily applied to this definition as well.

If we have several pieces of knowledge that are represented in the Dempster-Shafer form, then we can combine them into a single knowledge base. Several combination rules have been proposed. The original Dempster's rule is as follows: if we are given two pieces $(E_i, m_1(E_i))$ and $(F_i, m_2(F_i))$, then for the statements of the resulting knowledge base we take all consistent combinations $E_i \& F_j$, and to each of these statements X we assign the mass

$$m(X) = \frac{\sum_{i,j:\ X \leftrightarrow (E_i \& F_j)} m_1(E_i) m_2(F_j)}{\sum_{i,j:\ Consis(E_i \& F_j)} m(E_i) m(F_j)},$$

where $Consis$ means *consistent* and $\sum_{i,j:\ A}$ means the sum over all i, j, for which A is true.

Comment. Informally speaking, this means that we neglect all the inconsistent combinations and "divide" our belief between the consistent ones.

It was shown (Zadeh, 1984) that this rule sometimes contradicts our intuition. Following (Smets, 1988), let us briefly describe this contradiction. Suppose that we have three suspects in a murder case: Peter, Paul, and Mary, and two witnesses. The first witness is almost sure that Peter is a murderer, and his degrees of belief are: 0.99 that Peter murdered, 0.01 that Paul murdered, and 0 that Mary did it. The second witness has a 0.99 belief that Mary is the murderer, 0.01 that Paul is the murderer, and 0 that Peter is the one. From commonsense viewpoint, this means that we have strong suspicions against Peter and Mary. However, the original Dempster's rule leads to a different conclusion. Indeed, let us denote "Peter is the murderer" by E_1, "Paul is the murderer" by E_2, and "Mary is the murderer" by E_3. Then, the beliefs m_1, m_2 of the two witnesses are $m_1(E_1) = m_2(E_3) = 0.99, m_1(E_3) = m_2(E_1) = 0$, and $m_1(E_2) = m_2(E_2) = 0.01$. The original Dempster's combination rule than leads to $m(E_2) = 1$ and $m(E_1) = m(E_3) = 0$, i.e., to the conclusion that Paul is certainly the murderer.

Because of this contradiction, alternative combination rules were proposed by Yager (1985, 1987) and Smets (1988). Smets proposed the formula

$$m(X) = \sum_{i,j:\ X \leftrightarrow (E_i \& F_j)} m_1(E_i)m_2(F_j)$$

for all X (in particular for identically false $X = f$, that corresponds to the case, when the statements E_i and F_j are inconsistent). For the case when we have to combine $k > 2$ pieces of knowledge, he proposed a likewise formula

$$m(X) = \sum_{i,\ldots,j:\ X \leftrightarrow (E_i \& \ldots \& F_j)} m_1(E_i)\ldots m_k(F_j).$$

Yager applies this rule only for X, that are different from t ("identically true") and f ("identically false"), and assigns $m(f) = 0$ and

$$m(t) = \sum_{i,\ldots,j:\ \neg Consis(E_i \& \ldots \& F_j) \vee t \leftrightarrow (E_i \& \ldots \& F_j)} m_1(E_i)\ldots m_k(F_j).$$

Computational complexity is the main obstacle to the application of Dempster-Shafer formalism. Although this formalism is widely used, but there are some obstacles to its application, the main of which is its computational complexity (Bonissone, 1987, Dempster and Kong, 1987, Kyburg, 1987, Paass 1988, Pearl 1988, Hsia 1989, Phillips 1990). Indeed, when we apply one of the above combination rules to combine m pieces of knowledge, and each of them consist of at least 2 different statements, then we have to analyze at least 2^m different combinations of statements. Therefore we must make at least 2^m computational steps. For large m this is **infeasible** (e.g., for $m = 200$ it takes $> 10^{60}$ steps). So it is difficult to compute masses. But even if we manage to compute them, there is still a problem to compute beliefs from masses. If we directly apply the above formula for $bel(Q)$, and the number of statements in the resulting knowledge base is exponentially large, we must undertake exponentially many computational steps. The running time remains exponentially big even if we use the computationally optimal "fast Mobius" algorithm (Kennes, 1990, Kennes and Smets, 1990). So *what to do?*

For some cases faster algorithms are known that compute $bel(Q)$ for different Q in $< 2^m$ steps (Barnett, 1981), (Gordon and Shortliffe, 1985), (Shafer, 1985), (Shenoy and Shafer, 1986), (Shafer and Logan, 1987), (Wilson, 1989, 1991). These methods are applicable in many important cases, but still the problem remains: *what to do in the general case?*

11-4

General case: negative result. Orponen (1990) proved that exponential time is inevitable in the following sense: even for the propositional case the problem of computing beliefs is #P-complete (Garey and Johnson 1979). The majority of computer scientists believe that $P \neq NP$; so from their viewpoint no feasible algorithm is possible for computing beliefs (likewise results are proved in (Maung and Paris, 1990) for different uncertainty formalisms).

Is this negative result really tragic? To answer this question let's look at the usual logic, without any masses or degrees of belief. In this case knowledge consists of statements $E_1, ..., E_n$, and for every query Q the possible answers are "yes" (if $E_1 \& E_2 \& ... \& E_n \rightarrow Q$), "no" if $E_1 \& E_2 \& ... \& E_n \rightarrow \neg Q$ and "unknown" in all other cases. For this case many negative results are known, starting from the famous Godel's theorem. However, efficient inference engines and theorem provers exist and are successfully applied. In other words, theoretically the logical case is infeasible, but in practice it is feasible.

So the natural question is: is the Dempster-Shafer case practically feasible (in some reasonable sense) or not?

Feasible: in what sense? The natural formulation of this question is as follows: suppose that we already have an inference engine for logical statements, and we can use it as an additional tool while computing beliefs $bel(Q)$. In this case the running time is equal to the weighted sum of the number of real computational steps and the number of calls of this inference engine. Will this new computational time still be exponentially large?

If it is small, then we can quickly compute beliefs, and therefore DS approach is feasible. If this running time turns out to be exponentially large, this would mean that even the usage of the existing inference engines does not help, and therefore DS approach is infeasible.

The purpose of this paper is to analyze whether DS approach is practically feasible (in the above sense) or not. Our answer will be: "yes, it's feasible".

What we are planning to do. In Section 2 we give precise definitions and formulate a negative result: that the problem of computing beliefs precisely is practically infeasible. Since the initial masses express our degree of belief and are therefore only approximately known, there is no sense in trying to compute the beliefs precisely. However, as we show in Section 3, the problem of computing the beliefs with a given precision is also infeasible. In Section 4 we take into consideration that human experts can not only be slightly uncertain

1 - 5

about their degrees of belief, but can also have doubts in their whole knowledge. Therefore, since there is a probability that what an expert says is absolutely wrong, it is reasonable to allow the algorithms for computing $bel(Q)$ to err with some (very small) probability. For Dempster's rule the resulting problem is still infeasible, but for two other rules it is already feasible! In Section 5 we show that the methods from Section 4 can be parallelized, and what parallelization model fits this problem best. All the proofs are given in Section 6.

Our main results first appeared in (Borrett and Kreinovich 1990a, 1990b).

2. ALGORITHMS THAT COMPUTE BELIEFS PRECISELY ARE PRACTICALLY INFEASIBLE

Inference engine: general definition. Assume that some alphabet is given, that includes the symbols & and f; assume also that two sets of words form this alphabet are given. The words form the first set will be called *statements*, words from the second set *queries*. We assume that the word f (meaning *false*) belongs to both sets, and that if S_1 and S_2 are statements, then $S_1\&S_2$ is also a statement. Assume also that an algorithm I is given, that transforms every pair (S, Q), where S is a statement and Q is a query, into one of the words "*yes*" or "*no*". When $I(S, Q)=$"*yes*", we say that Q *follows from* S, or S *implies* Q and denote it by $S \to Q$. We say that the statements S_1 and S_2 are *equivalent* and denote it by $S_1 \leftrightarrow S_2$ if $S_1 \to S_2$ and $S_2 \to S_1$. We demand that this algorithm is *consistent* in the sense that if $A \leftrightarrow B$, then $I(A, Q) = I(B, Q)$ for all Q. Such consistent algorithms will be called *inference engines.*

Comment. One should bear in mind that in many cases (e.g., in first order logic) no algorithm is possible for which $I(S, Q)=$"*yes*" if and only if Q is a logical consequence of S. Therefore the notion "imply" that stems from the inference engine can be different from the logical implication.

We say that $E_1, ..., E_k$ *imply* Q if $E_1\&E_2\&...\&E_k$ implies Q. If S implies f, we say that S is *inconsistent*, else that S is *consistent*. The fact that a formula S is consistent will be denoted by $Consis(S)$.

Dempster-Shafer knowledge base: definition. By a *piece of knowledge* we mean a pair consisting of the finite set of statements $E_1, ..., E_n$ and a function m that assigns to each statement from this set a value $m(E_i) \geq 0$ so that $\sum m(E_i) = 1$. For every query Q we define the *belief* $bel(Q)$ in Q as the sum of $m(E_i)$ for all E_i, for which $I(E_i, Q)=$"*yes*" (i.e., for which E_i implies Q).

By a *Dempster-Shafer knowledge base* (or simply *knowledge base* for short) we mean a finite list of pieces of knowledge. For every knowledge base we can define the resulting piece of knowledge by applying one of the above-defined combination rules: Dempster's, Smets' and Yager's (we'll denote them by D, S and Y). For every query Q by a *belief* $bel(Q)$ in Q with respect to a knowledge base we mean its belief with respect to the resulting piece of knowledge.

Comment. In the combination formulas we must understand $\rightarrow$, $\leftrightarrow$ and $Consis$ in the sense of the inference engine I.

In the present section we'll consider algorithms that combine normal computational steps with calls of an inference engine I.

Comment. In more theoretical terms, we can say that we consider algorithms, that use I as an oracle (Garey and Johnson, 1979).

Assume that two positive real numbers are fixed: t_0 and t_c. t_0 will be called the *time of one computational step* and t_c *the time of one call*. For every input by a *running time* of an algorithm we mean the total number N_0 of normal computational steps, multiplied by t_0, plus the total number N_c of calls, multiplied by t_c. By the *length of the input* we mean the total length of the knowledge base and the query. By a *computational complexity* $t_U(n)$ of an algorithm U we mean the maximum of its running time on all the inputs of length $\leq n$.

We say that an algorithm *computes the beliefs precisely* if for every inference engine I, for every knowledge base and every query Q it computes $bel(Q)$.

THEOREM 1 (D, S, Y). *If an algorithm U computes the beliefs precisely, then $t_U(n) \geq ca^n$ for some $a > 1$.*

Comment. So, whatever combination rule we use, it takes exponentially many computational steps to compute beliefs precisely. Therefore the problem "to compute beliefs precisely" is infeasible.

3. ALGORITHMS THAT COMPUTE BELIEFS WITH GIVEN PRECISION ARE PRACTICALLY INFEASIBLE

Initial masses express our degree of belief. It is very difficult to express one's degree of belief with great precision: e.g., who can boast that he is 83% and not 84% sure in something? So these initial degrees of belief are only approximately known, and therefore there is no sense in trying to compute the resulting beliefs with bigger precision than the precision of the input data. So the natural idea is to fix some precision $\epsilon > 0$ and compute beliefs only with this precision. Alas, the resulting problem is also infeasible. Let us give precise definitions.

Definition. Assume that a positive number ϵ is fixed. We say that an algorithm U *computes the beliefs with precision* ϵ if for every inference engine I, for every knowledge base and every query Q it generates a real number $U(Q)$, for which $|U(Q) - bel(Q)| \leq \epsilon$.

Comment. If $\epsilon \geq 1/2$, then we can take an algorithm that always generates $1/2$, and thus satisfy this inequality for all possible values of belief. Therefore this definition makes sense only when $\epsilon < 1/2$.

THEOREM 2 (D, S, Y). *If an algorithm U computes the beliefs with precision $\epsilon < 1/2$, then $t_U(n) \geq ca^n$ for some $a > 1$.*

So this computation also demands exponential time and is therefore infeasible.

4. MONTE-CARLO METHODS: FEASIBLE FOR SMETS'S AND YAGER'S RULES, STILL INFEASIBLE FOR DEMPSTER'S RULE

Why probabilistic methods? Let us now take into consideration the fact that human experts can not only be slightly uncertain about their degrees of belief, but can also have doubts in their whole knowledge. In other words, there is a probability p_0 (small but positive) that what an expert says is absolutely wrong. If this is the case, then, no matter what algorithm we apply, the resulting values of belief will be absolutely inadequate. In view of that it is not necessary to achieve a 100% correctness of the algorithm. The only thing that is reasonable to demand is that the probability that an algorithm errs must be smaller than this p_0, so that the resulting probability of an error (due both to the possible errors of the algorithm and the errors in the initial data) is not much greater than p_0.

So we arrive at the following definitions:

Definitions. By a *standard random number generator* we mean a program or device that generates real numbers that are uniformly distributed on the interval [0,1]. By a *probabilistic algorithm* we mean an algorithm that in addition to normal computational steps and calling I calls a standard random number generator. The result of applying an algorithm U to the data x will be denoted by $U(x)$.

In addition to t_0 and t_c let us fix a number $t_r > 0$ (called *the time of one call of this random number generator*); let us define a *running time* of a probabilistic algorithm

11-8

U on any input data as $N_0 t_0 + N_c t_c + N_r t_r$, where N_0 is the total number of normal computational steps, N_c is the total number of calls of I, and N_r is the total number of calls of a generator. Let us now define the *computational complexity* $t_U(n)$ of an algorithm U as a maximum running time for all inputs of length n and for all possible values of the random number generator. If $t_U(n)$ is bounded by some polynomial of n, we call U a *polynomial-time algorithm*. If it is limited by a linear function, we call U a *linear-time* algorithm.

Comment. By definition a probabilistic algorithm uses a random number generator, therefore its output is not uniquely determined by the inputs: for every input it is a random variable.

Definition. Assume that positive numbers ϵ and p_0 are given. We say that a statement is *reliably true* if it is true with probability $1 - p_0$ or greater. We say that a probabilistic algorithm U *computes the beliefs with precision* ϵ *and reliability* $1 - p_0$ if for every knowledge base, for every inference engine I, and for every query Q it is reliably true that $|U(Q) - bel(Q)| \leq \epsilon$.

In other words, $P(|U(Q) - bel(Q)| \leq \epsilon) \geq 1 - p_0$.

Comment. In order to formulate the related negative result we must recall the denotation RP: it is the class of problem that can be solved (with reliability $1 - p_0$) in polynomial time. The majority of computer scientists believe that $RP \neq NP$ (for details see, e.g., Maung and Paris, 1990).

THEOREM 3 (D). *If $\epsilon < 1/4$, $p_0 < 1$, and $RP \neq NP$, then there is no polynomial-time algorithm that computes beliefs with precision ϵ and reliability $1 - p_0$.*

Comment. So if we use Dempster's combination rule, the problem of computing beliefs is still infeasible.

THEOREM 4 (S, Y) *For every $\epsilon < 1/4$ and $p_0 < 1$ there exists a linear-time algorithm that computes beliefs with precision ϵ and reliability $1 - p_0$.*

Comments. 1. **So the problem is feasible!**

2. Linear-time means that when the size of the problem increases (i.e., the number of pieces of knowledge increases, and/or the number of statements in every piece), then the number of calls of the inference engine I grows linearly of slower. What is this time equal

11 - 9

to in absolute units, that is, is it reasonably small or really big, depends on how quickly the inference engine works.

Description of the algorithm. Let us describe the algorithm from Theorem 4.

For that we need an auxiliary algorithm that given a piece of knowledge $(E_1, ..., E_n), m$, generates a statement E_i with probability $m(E_i)$. To get it we first compute the values $r_1 = m(E_1)$, $r_2 = m(E_1)+m(E_2)$, $r_3 = r_2+m(E_3)$, ..., $r_n = r_{n-1}+m(E_n) = m(E_1)+...+m(E_n) = 1$. Then we call a standard random number generator and compare the result r consequently with $r_1, r_2, ..., r_n = 1$. If $r \le r_1$, generate E_1; if $r_{i-1} < r \le r_i$, generate E_i.

Comments. 1. One can easily check that the probability of generating E_i is precisely $m(E_i)$.

2. This auxiliary algorithm is already a linear-time one. We can, however, further diminish its running time, if we use bisection search instead of a linear search.

As a second auxiliary step we must find an integer N depending on ϵ and p_0; in general case we can take $N = 2\epsilon^{-2}\ln(2/p_0)$. For small ϵ and p_0 we can take smaller values of N: e.g., to get a 10% precision and 95% reliability it is sufficient to take $N = 100$.

Now the main algorithm is as follows. Suppose that we are given a knowledge base, that consists of several pieces of knowledge $P_1, ..., P_k$. We do the following:

1) (*in case of Smets's combination rule*) Reserve an integer variable M for a counter, and set its initial value to 0. Then N time repeat the following:

Apply the auxiliary algorithm to each piece of knowledge P_i, and get a random statement E_i^r; then apply I to check whether Q follows from all these statements, i.e., whether $I(E_1^r \& E_2^r \& ... \& E_k^r, Q) =$"*yes*". If "*yes*", add 1 to the counter M; if not, leave M unchanged.

As a desired estimate for $bel(Q)$ we take M/N, where M is the value of the counter after N iterations.

2) (*in case of Yager's combination rule*) Same algorithm; the only difference is that we add 1 if $I(E_1^r \& E_2^r \& ... \& E_k^r, Q)=$"*yes*" and the set $\{E_1^r, E_2^r, ..., E_k^r\}$ is consistent, i.e., $I(E_1^r \& E_2^r \& ... \& E_k^r, f)=$"*no*".

Comments. 1. This algorithm belongs to the class of Monte-Carlo methods. Such methods were proposed for Dempster-Shafer formalism in (Pearl, 1988, Kampke, 1988, Laskey and Lerner, 1989). If we this algorithm with the original Dempster's combination rule, then for

11-10

the case when the conflicts between the pieces of knowledge are in some reasonable sense restricted, we also get linear-time estimates (Wilson, 1989, 1991).

2. Our result that Smets's and Yager's rules are better than Dempster's rule because they are feasible and Dempster's is not is in good accordance with the above-mentioned fact that the original Dempster's rule, unlike the two others, contradicts to our intuition (Zadeh, 1984).

5. PARALLEL COMPUTATION OF BELIEFS

Parallelization: possible advantages. Monte-Carlo methods can be easily implemented in parallel (see, e.g., Pearl, 1988): indeed, they consist of applying the inference engine to several randomly chosen sets of statements. If we have several processors at our disposal, then we can make each of them choose and process one set of statements. So each processor applies the inference engine only once, and the resulting running time of this parallel algorithm equals to the running time of the inference engine. So we compute the beliefs precisely in the same time as we apply the inference engine, and adding masses and beliefs does not increase the running time!

How to implement it. Theoretically the more processors we have, the quicker are the results. But in real parallel systems a lot of time is consumed on communication protocols, information exchange, waiting in the queues, etc. The more processors we have, the more time-consuming all these communication procedures become, and they seriously impact the whole computation process. So if we implement our parallelized algorithms on real parallel systems with many processors, this additional time will add to our running time and thus worsen our theoretical estimates.

In our case, however, during the main stage ("call inference engine") no communication is necessary, so we don't need to waste time on protocols. As a result each processor generates one bit ("yes" or "no"), and to estimate a belief we must send these bits to one processor and process them there. This can be done also without any protocols, by using a small shared memory of N bits, where N is the number of processors. Such an architecture was produced by Septor Electronics for use in machinery control applications (Roberts, 1989, Hardin and Taylor, 1990) and was efficiently used to parallelize Monte-Carlo algorithms (Kreinovich et al, 1990).

6. PROOFS

Comment. Some of the ideas that we use in these proofs appeared first in (Dantsin, 1990; Dantsin and Kreinovich, 1990).

Proofs of Theorems 1 and 2. If an algorithm computes beliefs precisely, then, of course, it also computes beliefs with precision ϵ. Therefore, if we prove Theorem 2, we get Theorem 1 as a corollary. So it is sufficient to prove Theorem 2. Let's do it.

Since we are proving a negative result, it is sufficient to construct a case, in which the algorithm must work for a long time. Suppose that U is an algorithm that computes beliefs with precision ϵ. Let's take a knowledge base that consists of n pieces of knowledge $P_1, P_2, ..., P_n$. i^{th} piece of knowledge consists of two statements E_i and $\neg E_i$, with $m_i(E_i) = m_i(\neg E_i) = 0.5$. For Q let us take a statement that is different from any Boolean combination of these E_i. We'll use the denotations E_i^+ for E_i and E_i^- for $\neg E_i$.

Let us consider only such I, that for Boolean formulas, formed from E_i, coincide with logical implication. So, for example, $E_1 \& E_2 \rightarrow E_1$, but $E_1 \& \neg E_2 \not\rightarrow E_3$. In particular, for every sequence $\vec{\epsilon} = (\epsilon_1, ..., \epsilon_n)$ of + and − symbols $I(E_1^{\epsilon_1} \& E_2^{\epsilon_2} \& ... \& E_n^{\epsilon_n}, f)$= "no". In other words, all possible combinations of E_i and $\neg E_i$ are consistent.

Three combination rules differ only in case of inconsistent knowledge. Therefore for the above-described case, when all the combinations are consistent, all three rules lead to the same combined knowledge. This knowledge consists of 2^n statements $E = E_1^{\epsilon_1} \& E_2^{\epsilon_2} \& ... \& E_n^{\epsilon_n}$, and the mass of each statement equals to $m_1(E_1^{\epsilon_1}) m_2(E_2^{\epsilon_2}) ... m(E_n^{\epsilon_n}) = (1/2)^n = 2^{-n}$. So for every query Q the belief $bel(Q)$ equals to the sum of the masses of all the statements that imply Q, i.e., to $2^{-n} N(Q)$, where by $N(Q)$ we denoted the total number of statements E that imply Q.

Let us denote the number of times, during which our algorithm U called the inference engine I to know whether E implies Q for some E, by N. If $N < 2^n$, this means that for some of 2^n combinations E we did not ask whether $E \rightarrow Q$. So, if we take an inference engine J that coincides with I on all Boolean combinations of E_i and on all the pairs (E, Q), for which the algorithm U called I, and apply the same algorithm to this J instead of I, U will not feel the difference, because whenever it asks an inference engine something, it still gets the same results. So the result $U_J(Q)$ of applying this algorithm to J will be same, as in case of I: $U_J(Q) = U_I(Q)$. Let's take two such J: J_1 says "yes" for all pairs (E, Q), for which U did not ask I; and J_2 answers "no" on all such pairs. Let us

11-12

denote the number of statements, for which $J_i(E,Q)$="yes", by $N_i(Q)$. The difference between $N_1(Q)$ and $N_2(Q)$ consists precisely of $2^n - N_c$ statements E, for which U did not ask I, i.e., $N_1(Q) - N_2(Q) = 2^n - N$. Since in our case $bel(Q) = 2^{-n}N(Q)$, we conclude, that the values of belief $bel_i(Q)$, that correspond to J_i, satisfy the equality $bel_1(Q) - bel_2(Q) = 2^{-n}(2^n - N) = 1 - 2^{-n}N$. But we took an algorithm that computes beliefs with precision ϵ, therefore the result $U(Q)$ of this algorithm must differ from both of these beliefs by no more than ϵ: $|bel_1(Q)-U(Q)| \leq \epsilon$ and $|bel_2(Q)-U(Q)| \leq \epsilon$. From these inequalities we conclude, that $|bel_1(Q)-bel_2(Q)| \leq |bel_1(Q)-U(Q)|+|bel_2(Q)-U(Q)| \leq 2\epsilon$, and so $1 - 2^{-n}N \leq 2\epsilon$. So $2^{-n}N \geq (1 - 2\epsilon)$ and therefore $N \geq 2^n(1 - 2\epsilon)$. Since $\epsilon < 1/2$, this difference $1 - 2\epsilon$ is positive.

So the total running time is $\geq t_cN \geq 2^n(1-2\epsilon)$, i.e. it is really exponentially increasing with the length of the input. Q.E.D.

Proof of Theorem 3. Let's prove this theorem by reductio ad absurdum: we'll suppose that such a polynomial algorithm U exists, and conclude that $RP = NP$, i.e., that there exists a polynomial-time probabilistic algorithm that solves one of NP-complete problems. Namely, we'll construct such an algorithm for the propositional satisfiability problem. Indeed, suppose that U exists, and we have a propositional formula P with n propositional variables $x_1, ..., x_n$. Let's figure out whether this formula is satisfiable or not. For that purpose let's introduce a new propositional variable x_{n+1} and consider the knowledge base that consists of the following $n+1$ pieces of knowledge $P_1, ..., P_{k+1}$. When $i \leq k$, then P_i consists of two statements x_i and $\neg x_i$ with equal masses. P_{k+1} consists of two statements $P\&\neg x_{n+1}$ and $x_1\&x_2\&...\&x_n\&x_{n+1}$, also with equal masses.

Statements of the combined knowledge base are formed as follows: for every i from 1 to n we must choose either x_i or $\neg x_i$, and then we must choose either $P\&\neg x_{n+1}$, or $x_1\&x_2\&...\&x_n\&x_{n+1}$. In other words, we must first choose an n-dimensional Boolean vector $\vec{x} = (x_1, ..., x_n)$, and then choose one of the statements, with which it is consistent: $P\&\neg x_{n+1}$ or $x_1\&x_2\&...\&x_n\&x_{n+1}$. For each vector $\vec{x}$ consistency is easy to check: if P is true for this $\vec{x}$ (and this can be checked in polynomial time), then the first is consistent, if $x_1 = x_2 = ... = x_n =$ "*true*", then the second one is consistent. So we can easily implement logical consistency checking for these cases.

The resulting masses are as follows: If P is not satisfiable, then the combinations with $P\&\neg x_{n+1}$ are inconsistent, and therefore the only consistent combination is $x_1\&x_2\&...\&x_n\&x_{n+1}$; therefore it gets the mass 1. If P is satisfiable, and N is the number of Boolean vectors that satisfy it, then we have $N+1$ consistent combinations. Since all

(11-13)

the masses in all the pieces of knowledge are equal, the masses assigned to these consistent combinations are also equal, so we assign $1/(N+1)$ to each of them.

Let us take $Q = x_{n+1}$. For every statement E from the combined knowledge base we already know x_{n+1}, so the trivial algorithm will work as I in this case. The resulting belief $bel(Q)$ is as follows: if P is satisfiable, then $bel(Q) = 1$. If P is not satisfiable, then $bel(Q) = 1/(N+1)$, where $N \geq 1$, so $bel(Q) \leq 1/2$.

If $|U(Q) - bel(Q)| < 1/4$, then in case $bel(Q) = 1$ we have $U(Q) > 1 - 1/4 = 3/4$, and in case $bel(Q) \leq 1/2$ we have $U(Q) < 1/2 + 1/4 = 3/4$. So if we apply U to this knowledge base and compare $U(Q)$ with 3/4, we can tell whether P is satisfiable or not: if $U(Q) < 3/4$, it is satisfiable; when $U(Q) > 3/4$, it is not. So, using U, we constructed an algorithm that checks whether a formula is satisfiable with reliability $\geq 1 - p_0$. So our assumption that a polynomial-time algorithm U can compute beliefs with given precision and reliability contradicts to the assumption that $RP \neq NP$. Therefore such an algorithm U is impossible. Q.E.D.

Proof of Theorem 4. That the algorithm described in Section 4 is linear-time can be easily seen: the number N_c of calls for I equals either to N (in Smets's case) or to $2N$ (in Yager's case); in both cases it does not depend on the input length at all. Likewise the number of times during which this algorithm calls the random-number generator is limited by N. As for additional computations, for each of N iterations they demand looking through all the pieces of the knowledge base N times, i.e., the necessary running time is $\leq const \cdot Nn$ and is therefore linear in n.

Let us now prove that these linear-time algorithms really work. Let's first consider the Smets's rule. By definition $bel(Q) = \sum_{E:\ E \to Q} m(E)$, where E runs over all combinations $E_i \& ... \& F_j$ of statements form different pieces of knowledge. And the masses $m(E)$ are equal to $m(E) = \sum_{i,j:\ E \leftrightarrow (E_i \& ... \& F_j)} m_1(E_i) ... m_k(F_j)$. Substituting this expression for $m(E)$ into the formula for $bel(Q)$, we conclude, that $bel(Q) = \sum_{E:\ E \to Q} \sum_{i,j:\ E \leftrightarrow (E_i \& ... \& F_j)} m_1(E_i) ... m_k(F_j)$. We defined an inference engine as a consistent algorithm, i.e., an algorithm, for which $A \leftrightarrow B$ implies that $I(A, Q) = I(B, Q)$. In particular, if $E \leftrightarrow (E_i \& ... \& F_j)$, then $E \to Q$ if and only if $(E_i \& ... \& F_j) \to Q$. Therefore the above expression for $bel(Q)$ can be simplified:

$$bel(Q) = \sum_{i,j:\ (E_i \& ... \& F_j) \to Q} m_1(E_i) ... m_k(F_j).$$

Let us now prove that this expression equals to some probability. We say that a statement is *randomly chosen* from a piece of knowledge $((E_1, ..., E_k), m)$ if it coincides with E_i with probability $m(E_i)$. We suppose that the choices from different pieces of knowledge are independent. In this case the probability that a sequence $E_i, ..., F_j$ is chosen, equals to $m_1(E_i)...m_k(F_j)$. Therefore the right-hand side of the above formula for $bel(Q)$ is the sum of the probabilities of all cases, in which the chosen sequence implies Q, i.e., $bel(Q)$ equals to the probability that a random sequence implies Q.

Comment. This fact does not mean that we interpret masses as probabilities: it is a purely formal equality, that may have nothing to do with semantics of masses, but that turns out to be useful for computing beliefs.

This probability can be computed as follows: we make several (N) simulations of the random event, and estimate probability p by a ratio M/N, where M is the number of cases, in which the event happened (in our case in which the randomly chosen sequence implied Q). The precision of these estimates is known from mathematical statistics: to get a precision ϵ with reliability $1 - p_0$, we must take $N = 2\epsilon^{-2}\ln(2/p_0)$ (so called Hoefding theorem; see also Dantsin and Kreinovich, 1990, Wilson, 1989, 1991).

For big N the distribution for the difference $p - M/N$ is close to Gaussian, so we can use the estimates for the Gaussian distribution. In particular, we get $N = 100$ for $\epsilon = 0.1$ and $p_0 = 0.05$.

For Yager's case the arguments are the same, with the only difference that $bel(Q)$ is equal to the probability that the randomly chosen sequence is consistent and implies Q. Q.E.D.

ACKNOWLEDGEMENTS

This research was supported by NSF grant No. CDA-9015006, NASA Research grant NAG 9-482, a grant from the Institute for Manufacturing and Materials Management and an equipment donation from Septor Electronics. The authors are also thankful to E. Dantsin (Leningrad), D. Dubois (Toulouse), M. Gelfond (El Paso), J. Paris (Manchester), J. Pearl (Los Angeles), H. Prade (Toulouse), Ph. Smets (Brussels) and P. Suppes (Stanford) for valuable discussions.

11-15

REFERENCES

Barnett J.A. (1981) *Computational methods for a mathematical theory of evidence* in Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI-81), Vancouver, pp. 868–875.

Bonissone P. P. (1987) *Reasoning, Plausible* in S. C. Shapiro (Ed.) *Encyclopedia of Artificial Intelligence*, John Wiley, N. Y.

Borrett W. and V. Kreinovich (1990a) *Monte-Carlo methods allow to avoid exponential time in Dempster-Shafer formalism.* Technical Report UTEP-CS-90-05, University of Texas at El Paso.

Borrett W. and V. Kreinovich (1990b) *Monte-Carlo methods allow to avoid exponential time in Dempster-Shafer formalism.* Abstracts Amer. Math. Soc. 11, p. 473.

Dantsin E. (1990) *Algorithms for probabilistic inference* in Springer Lecture Notes in Computer Science 417, pp. 67–75.

Dantsin E. and V. Kreinovich (1990) *Probabilistic inference in prediction systems.* Soviet Math. Doklady 40, pp. 8- 12.

Dempster A. P. and A. Kong (1987) a *discussion of* G. Shafer *Probability judgement in Artificial Intelligence and expert systems.* Statistical Science 2, pp. 3–44.

Dubois D. and H. Prade (1988) *Representation and combination of uncertainty with belief functions and possibility measures* Computational Intelligence 4, pp. 244–264.

Garey M. R. and D. S. Johnson (1979) *Computers and intractability* - *a guide to the theory of $\mathcal{NP}$-completeness*, Freeman, N.Y.

Gordon J. and E. H. Shortliffe (1985) *A method for managing evidential reasoning in a hierarchical hypothesis space.* Artificial Intelligence 26, pp. 323–357.

Hardin J. and J. Taylor (1990) *A distributed parallel processing system can solve today's complex automated machine control problems*, in Proceedings 19th Annual International Programmable Controllers Conference.

Hsia Y.-T. (1989) *Valuation invariance, epistemic irrelevance and the use of the Dempster-Shafer theory for reasoning.* Universite Libre de Bruxelles, IRIDIA Technical Report No. 89-9.

11-16

Kampke T. (1988) *About assessing and evaluating uncertain inferences within the theory of inference.* Decision Support Systems 4, pp. 433–439.

Kennes R. (1990) *Computational aspects of the Mobius transform of a graph*, Universite Libre de Bruxelles, IRIDIA Technical Report No. 90-13.

Kennes R. and P. Smets (1990) *Computational aspects of the Mobius transform*, in Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, Cambridge, MA, pp. 344- 351.

Kreinovich V., A. Bernat, E. Villa and Y. Mariscal (1991) *Parallel computers estimate errors caused by imprecise data.* Proc. Fourth ISMM (International Society on Mini and Micro Computers) International Conference on Parallel and Distributed Computing and Systems, Washington, Vol. 1, pp. 386–390.

Kyburg H. E. Jr. (1987) *Bayesian and non-bayesian evidential updating.* Artificial Intelligence 31, pp. 271–293.

Laskey K. B. and P. E. Lerner (1989) *Assumptions, beliefs and probabilities.* Artificial Intelligence 41, pp. 65–77.

Maung I. and J. B. Paris (1990) *A note on the infeasibility of some inference processes.* International Journal of Intelligent Systems 5, pp. 595–603.

Orponen P. (1990) *Dempster's rule of combination is #P-complete.* Artificial Intelligence 44, pp. 245–253.

Paass G. (1988) *A discussion on Dempster-Shafer formalism*, in (Smets et al, 1988), pp. 279–280.

Pearl J. (1988) *Probabilistic reasoning in intelligent systems*, Morgan Kaufmann, San Mateo, CA.

Phillips D. (1990) *Belief maintenance using the Dempster-Shafer theory of evidence.* The C Users Journal, No. 3, pp. 67–78.

Roberts R. (1989) *AI enhanced transfer lines.* Programmable Controls, May, pp. 105–108.

Shafer G. (1976) *A mathematical theory of evidence.* Princeton University Press, Princeton.

Shafer G. (1985) *Hierarchical evidence* in Proceedings of the Second Conference on Artificial Intelligence Applications, IEEE Press, pp. 16–21.

11-17

Shafer G. and R. Logan (1987) *Implementing Dempster's rule for hierarchical evidence.* Artificial Intelligence 33, pp. 271–298.

Shenoy P. P. and G. Shafer (1986) *Propagating belief functions with local computations.* IEEE Expert 1, pp. 43–52.

Smets P. (1988) *Belief functions* in (Smets et al, 1988), pp. 253–286.

Smets P. et al, Eds. (1988) *Nonstandard logics for automated reasoning,* Academic Press, London.

Wilson N. (1989) *Justification, computational efficiency and generalization of the Dempster-Shafer theory,* Oxford Polytechnic Dept. of Computing and Math. Sci. Technical Report No. 15, 1989.

Wilson N. (1991) *A Monte-Carlo algorithm for Dempster-Shafer belief.* in Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence, Los Angeles, CA.

Yager R. R. (1985) *On the relationships of methods of aggregation evidence in expert systems.* Cybernetics and Systems 16, pp. 1–21.

Yager R.R. (1987) *On the Dempster-Shafer framework and new combination rules.* Information Sciences 41, pp. 93–137.

Zadeh L. (1984) *A mathematical theory of evidence.* AI Magazine 5, pp. 81–83.

11 – 18